Manual de usuario

Analizador Léxico

Lenguaje NESP

Compiladores

Integrantes:

Daniel Eduardo Ruiz C.

22 de Marzo, 2025

Periodo 2025a

Introducción

Para el análisis léxico o conversión de código fuente a tokens de un lenguaje de programación en español que pretende tener coherencia lógica y ser más cercano al lenguaje natural, así como estar al mismo tiempo en español para que sea más amigable con los usuario nuevos. En el ánimo de crear un compilador para el mismo usando ANTLR en su versión para python.

Descripción

NESP, este lenguaje se basa en utilizar diversas expresiones estructuras en español con relativa facilidad y semejanza a oraciones simples que pretende emular la fase de desarrollo cuando se estructura un algoritmo, reemplazando palabras clave y operadores tradicionales por términos en español, con el objetivo de facilitar el razonamiento.

Componentes principales:

- gramaticaEspanol.g4: Gramática del analizador léxico.
- analizador_lexico.py: Script de python para usar el analizador.

Instalación y uso

1. Requisitos
    a. Python >= 3.6
    b. ANTLR4 Runtime para python.

2. Ejecutar el siguiente script:

```
pip install antlr4-tools
pip install antlr4-python3-runtime
antlr4 -Dlanguage=Python3 GramaticaEspanol.g4
```

3. Para analizar un archivo.

```
python analizador_lexico.py ejemplo.nesp
```

Características

Reglas generales:

1. Las funciones deben comenzar con funcion y finalizar con fin.
2. Los bloques condicionales (si) deben llevar entonces: y cerrarse con fin.
3. Los bucles (mientras) deben llevar entonces: y cerrarse con fin.

Palabras Reservadas:

- si ➜ Condicional (equivalente a if).
- entonces: ➜ Inicia bloque condicional y siempre debe ir seguido de :.
- fin ➜ Indica el fin de un bloque (} en otros lenguajes).
- mientras ➜ Bucle (while).
- asigna ➜ Operador de asignación (=).
- a la ➜ Operador de potencia (^).
- devuelve ➜ Retorno de función (return).

- funcion ➜ Definición de función (function).

- es un ➜ Declaración de tipo de dato.

Estructuras de Control:

- si condicion entonces: ➜ Inicia un bloque condicional.
- fin ➜ Indica el fin del bloque.
- mientras condición entonces: ➜ Inicia un bucle, finaliza con fin.

Definición de Funciones:

- Se declaran con funcion, seguidas del nombre y los parámetros entre paréntesis.
- El tipo de retorno se define con devuelve.
- El cuerpo de la función comienza después de : y termina con fin.

Tipos de Datos:

- entero ➜ Números enteros (int).
- real ➜ Números con decimales (float).

- texto ➜ Cadenas de caracteres (string).
- bit ➜ Booleano (true/false).
- arreglo de ➜ Estructuras multidimensionales (array).

Operadores Relacionales:

- es distinto de ➜ !=
- es igual que ➜ ==
- es mayor que ➜ >
- es menor que ➜ <
- es mayor o igual que ➜ >=
- es menor o igual que ➜ <=

Operadores Aritméticos:

- mas ➜ +
- menos ➜ -

- por ➜ *
- entre ➜ /

Operadores Lógicos:

- y ➜ &&
- o ➜ ||
- no ➜ !

Identificadores:

- Inician con una letra y pueden contener letras, números y guiones bajos (_).

Constantes y Literales:

- Números enteros y reales: 123, 3.14, -25.6
- Cadenas: Texto delimitado por comillas ("Hola mundo").

Delimitadores y Símbolos Especiales:

- (y) ➜ Paréntesis

- : ➜ Obligatorio después de entonces: y después de definir una función.
- . ➜ Fin de línea (equivalente a ;).
- fin ➜ Indica el fin de una estructura ({} en otros lenguajes).

Manejo de Espacios, Indentaciones y Comentarios:

- Los comentarios deben comenzar con Nota: y terminar con /.
- Se ignorar espacios e indentaciones repetidas.

Tabla de expresiones regulares

ALGUNOS TOKENS VARÍAN AL SER IMPLEMENTADOS.

Token	Expresión Regular	Ejemplo
FUNCION	funcion ID\((ID,)*ID \): (INST)* FIN	funcion sumar(a, b):
SI	si OP_LOG ENTONCES (INST)* FIN	si a igual que b entonces:
ENTONCES	entonces:	si x menor que 5 entonces:
FIN	fin	fin
LLAMA_FUNC	ID\(((ID | NUM | CADENA | 0 | 1),)*(ID | NUM | CADENA | 0 | 1) \)	hola(a,h,b,100)

Token	Expresión Regular	Ejemplo
INST	(TIPO_DATO | ASIGNA | DEVUELVE | LLAMA_FUNC . | SI | MIENTRAS | FUNCION)	a es un entero. hola(a,h,b,100).
MIENTRAS	mientras OP_LOG ENTONCES (INST)* FIN	mientras i menor que 10 entonces:
ASIGNA	ID asigna (OP_ARIT | OP_REL | OP_LOG) .	c asigna a mas b
A_LA	(ID | NUMERO) a la (ID | NUMERO)	2 a la 3
DEVUELVE	devuelve (ID | OP_ARIT | OP_REL | OP_LOG) .	devuelve resultado
OP_REL	(ID | NUMERO | OP_ARIT | OP_REL | LLAMA_FUNC) es (distinto de | igual que | mayor que | menor que | mayor o igual que | menor o igual que) (ID | NUMERO | OP_ARIT | OP_REL | LLAMA_FUNC) | \(OP_REL \)	x distinto de y
OP_ARIT	((ID | NUMERO | OP_ARIT | LLAMA_FUNC) (mas | menos | por | entre | A_LA) (ID | NUMERO | OP_ARIT | LLAMA_FUNC))* | \(OP_ARIT \)	a mas b
OP_LOG	(ID|0|1| OP_REL | OP_LOG | LLAMA_FUNC) (y | o | no) (ID|0|1| OP_REL | OP_LOG | LLAMA_FUNC)* | \(OP_LOG \)	a y b
TIPO_DATO	ID es un (entero | real | texto | bit | (arreglo de (entero | real | texto | bit) (DE ENTERO)*)) .	nombre es texto
ID	[a-zA-Z][a-zA-Z0-9_]*	variable1

Token	Expresión Regular	Ejemplo
NUMERO	\d+(\.\d+)?	3.14
CADENA	"([^"\n]*)"	"Hola mundo"
COMENTARIO	Nota: .*?/	Nota: Esto es un comentario./
PARENTESIS	[\(\)]	(expresion)
DOS_PUNTOS	:	entonces:
PUNTO	\.	fin de instrucción.

Ejemplo

```
Nota: Este es un programa de ejemplo para probar el analizador léxico y sintáctico./

funcion factorial(n):
    si n es menor o igual que 1 entonces:
        devuelve 1.
    fin
    devuelve n por factorial(n menos 1).
fin

Nota: Ejemplo de declaración de variables y arreglos./
contador es un entero.
temperatura es un real.
nombre es un texto.
activo es un bit.
```

```
Nota: Declaración de arreglos según el formato especificado./
vector es un arreglo de entero de 10.
matriz es un arreglo de real de 3 por 3.
cubo es un arreglo de entero de 2 por 2 por 2.

Nota: Ejemplo de asignación y operaciones./
contador asigna 1.
mientras contador es menor que 10 entonces:
    contador asigna contador mas 1.
    si contador es igual que 5 entonces:
        Nota: Esto es un comentario dentro del bloque./
        temperatura asigna 25.5.
    fin
fin

resultado asigna 2 a la 3.
nombre asigna "Hola Mundo".
```

Conclusión

Se desarrollarán los próximos análisis partiendo de este primero además sería interesante agregar algún paradigma como programación orientada a objetos o a prototipos, funcional , etc. Pero por el momento sólo hay funciones, no se pueden definir objetos ni clases.